



**Faculty of Engineering & Technology**  
**Electrical & Computer Engineering Department**

**ENEE33203**

**COMPUTER NETWORKS Project #1**

---

Student Name : Shahd Yahya

Student ID : 1210249

Instructor: Dr.Abdalkarim Awad

Section: 2

## Part 1:

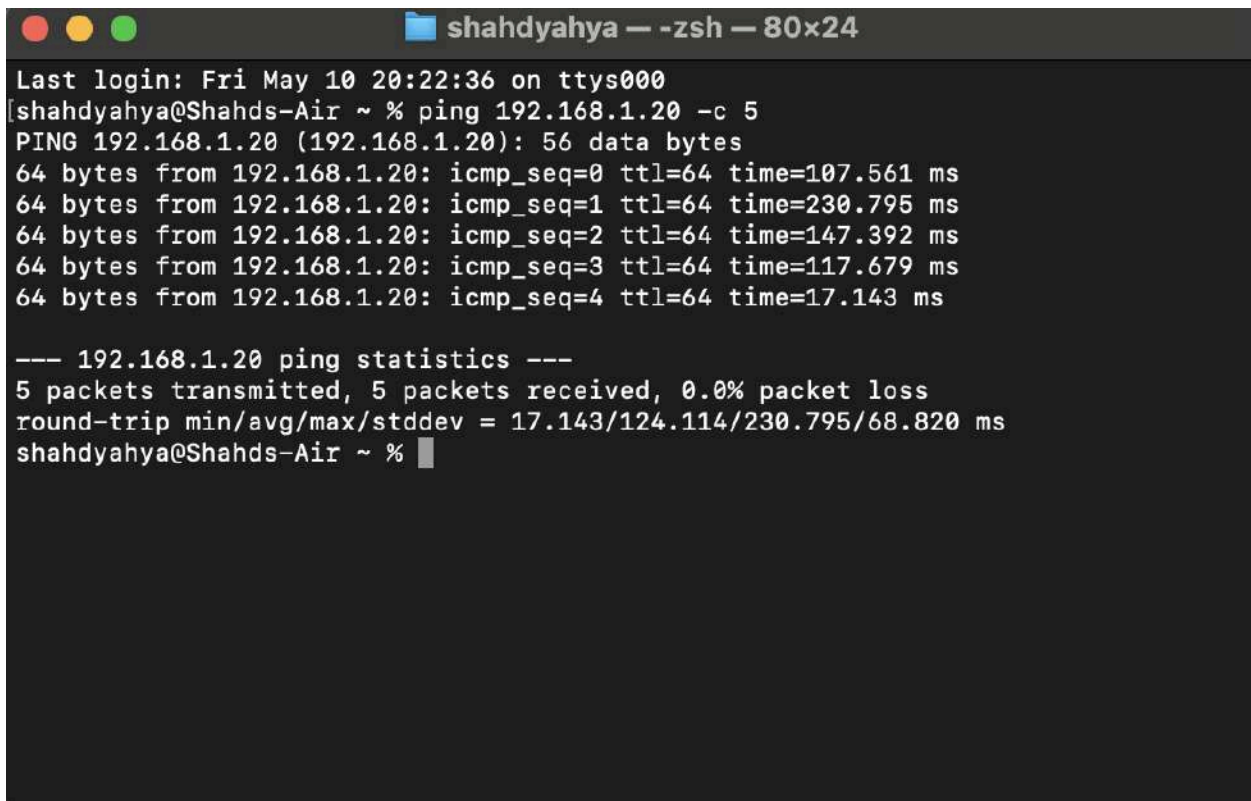
1. **Ping:** It's a simple utility that sends a data packet to another device on the network and waits for a response. This helps check if the device is reachable and how long it takes for the signal to go back and forth (round-trip time).

**Tracert:** It tracks the route data packets take from your computer to a destination, showing each "hop" (router) along the way. This can help identify bottlenecks or connection issues along the path.

**Nslookup:** It's a tool used to query Domain Name System (DNS) servers. You can use it to translate website addresses (domain names) into numerical IP addresses that computers use, or look up other DNS records.

2.

Ping my phone from my laptop:

A terminal window titled 'shahdyahya — -zsh — 80x24' with standard macOS window controls (red, yellow, green buttons). The terminal shows the output of a ping command. It starts with 'Last login: Fri May 10 20:22:36 on ttys000'. The user enters '[shahdyahya@Shahds-Air ~ % ping 192.168.1.20 -c 5]'. The output shows 'PING 192.168.1.20 (192.168.1.20): 56 data bytes' followed by five lines of packet details: '64 bytes from 192.168.1.20: icmp\_seq=0 ttl=64 time=107.561 ms', '64 bytes from 192.168.1.20: icmp\_seq=1 ttl=64 time=230.795 ms', '64 bytes from 192.168.1.20: icmp\_seq=2 ttl=64 time=147.392 ms', '64 bytes from 192.168.1.20: icmp\_seq=3 ttl=64 time=117.679 ms', and '64 bytes from 192.168.1.20: icmp\_seq=4 ttl=64 time=17.143 ms'. Below this is a summary: '--- 192.168.1.20 ping statistics ---', '5 packets transmitted, 5 packets received, 0.0% packet loss', and 'round-trip min/avg/max/stddev = 17.143/124.114/230.795/68.820 ms'. The prompt 'shahdyahya@Shahds-Air ~ %' is shown at the bottom with a cursor.

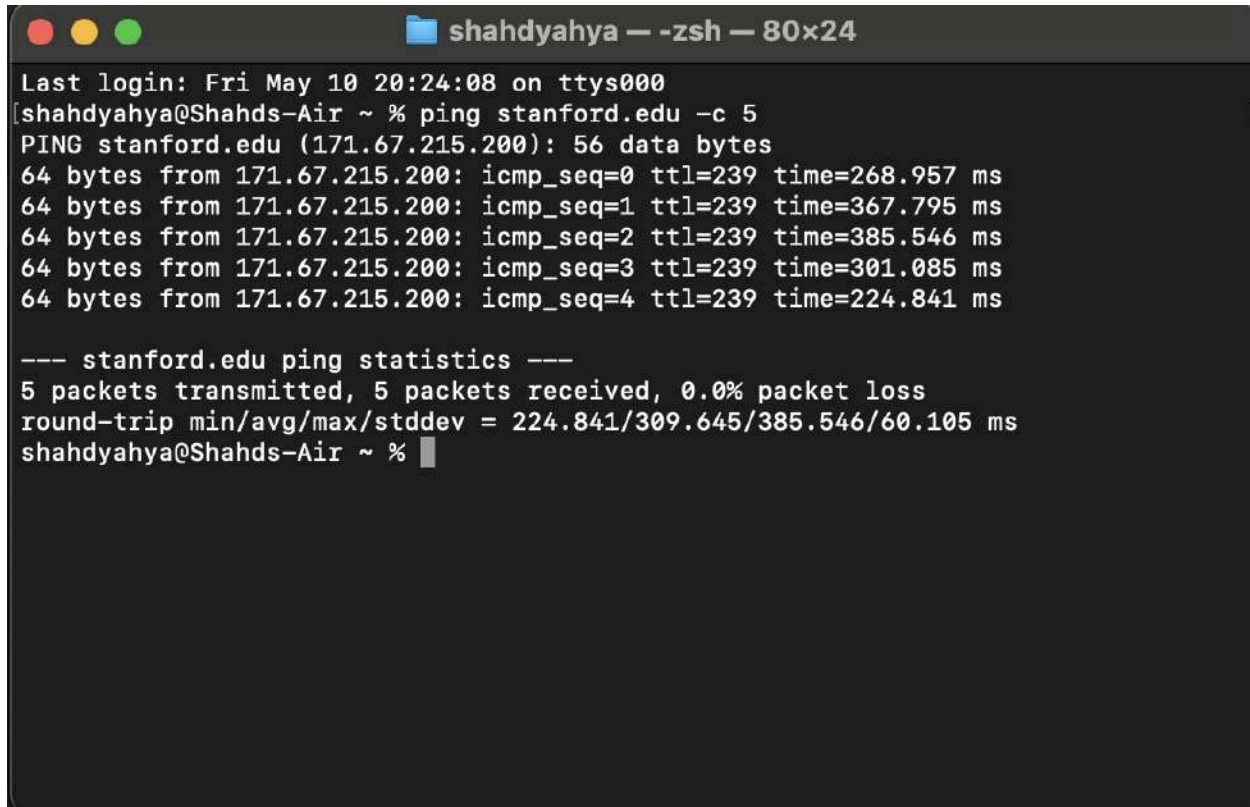
```
Last login: Fri May 10 20:22:36 on ttys000
[shahdyahya@Shahds-Air ~ % ping 192.168.1.20 -c 5
PING 192.168.1.20 (192.168.1.20): 56 data bytes
64 bytes from 192.168.1.20: icmp_seq=0 ttl=64 time=107.561 ms
64 bytes from 192.168.1.20: icmp_seq=1 ttl=64 time=230.795 ms
64 bytes from 192.168.1.20: icmp_seq=2 ttl=64 time=147.392 ms
64 bytes from 192.168.1.20: icmp_seq=3 ttl=64 time=117.679 ms
64 bytes from 192.168.1.20: icmp_seq=4 ttl=64 time=17.143 ms

--- 192.168.1.20 ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 17.143/124.114/230.795/68.820 ms
shahdyahya@Shahds-Air ~ %
```

I have transmitted 5 packets to my phone (in the same network) and received 5 packets with 0% packet loss: All 5 ping requests were sent and successfully received, meaning there were no dropped packets. Round-trip time (RTT) is the total time it takes for a ping request to reach the device and return a response. Here,

the RTT ranged from a minimum of 17.143 milliseconds to a maximum of 230.795 ms, with an average of 124.114 ms.

Ping stanford.edu:

A terminal window titled 'shahdyahya — -zsh — 80x24' showing the output of a ping command. The window has a dark background with light-colored text. The output shows five successful ping requests to stanford.edu (171.67.215.200) with varying round-trip times. A summary line indicates 5 packets transmitted, 5 received, and 0% packet loss, with a round-trip time range of 224.841 to 385.546 ms and an average of 309.645 ms.

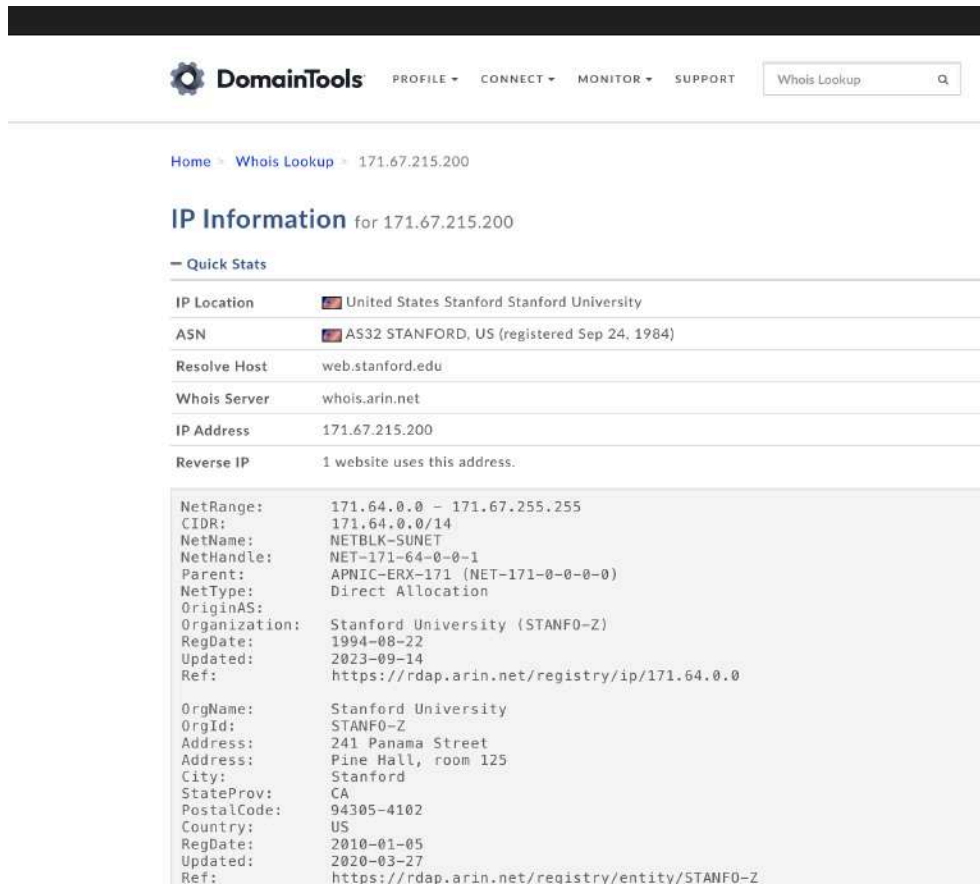
```
Last login: Fri May 10 20:24:08 on ttys000
[shahdyahya@Shahds-Air ~ % ping stanford.edu -c 5
PING stanford.edu (171.67.215.200): 56 data bytes
64 bytes from 171.67.215.200: icmp_seq=0 ttl=239 time=268.957 ms
64 bytes from 171.67.215.200: icmp_seq=1 ttl=239 time=367.795 ms
64 bytes from 171.67.215.200: icmp_seq=2 ttl=239 time=385.546 ms
64 bytes from 171.67.215.200: icmp_seq=3 ttl=239 time=301.085 ms
64 bytes from 171.67.215.200: icmp_seq=4 ttl=239 time=224.841 ms

--- stanford.edu ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 224.841/309.645/385.546/60.105 ms
shahdyahya@Shahds-Air ~ %
```

I have transmitted 5 packets and received 5 packets with 0% packet loss: All 5 ping requests were sent and successfully received, meaning there were no dropped packets. Round-trip time (RTT) is the total time it takes for a ping request to reach the device and return a response. Here, the RTT ranged from a minimum of 224.841 ms to a maximum of 385.546 ms, with an average of 309.645 ms.

I have noticed that pinging a device in the same network is faster and that is because of the distance and congestion

To test if the ping result is from America I put the IP address that I got(171.67.215.200) on Domain Tool on the internet and it showed that the location is America



The screenshot shows the DomainTools website interface. At the top, there's a navigation bar with 'DomainTools' logo and links for PROFILE, CONNECT, MONITOR, and SUPPORT. A search bar on the right contains 'Whois Lookup' and a magnifying glass icon. Below the navigation bar, a breadcrumb trail reads 'Home > Whois Lookup > 171.67.215.200'. The main heading is 'IP Information for 171.67.215.200'. Underneath, there's a 'Quick Stats' section with a table of key information:

IP Location	United States Stanford Stanford University
ASN	AS32 STANFORD, US (registered Sep 24, 1984)
Resolve Host	web.stanford.edu
Whois Server	whois.arin.net
IP Address	171.67.215.200
Reverse IP	1 website uses this address.

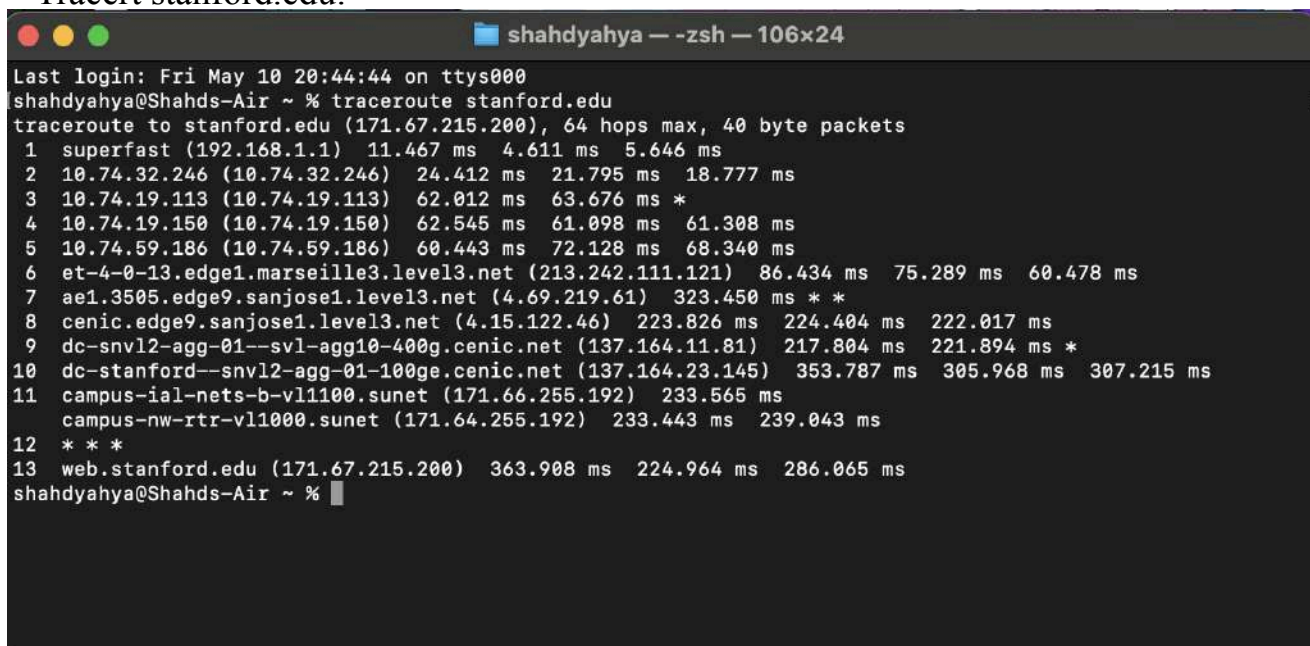
Below the table is a detailed 'NetRange' section with the following data:

NetRange: 171.64.0.0 - 171.67.255.255  
CIDR: 171.64.0.0/14  
NetName: NETBLK-SUNET  
NetHandle: NET-171-64-0-0-1  
Parent: APNIC-ERX-171 (NET-171-0-0-0-0)  
NetType: Direct Allocation  
OriginAS: Stanford University (STANF0-Z)  
Organization: Stanford University (STANF0-Z)  
RegDate: 1994-08-22  
Updated: 2023-09-14  
Ref: https://rdap.arin.net/registry/ip/171.64.0.0

Below this is an 'OrgName' section with the following data:

OrgName: Stanford University  
OrgId: STANF0-Z  
Address: 241 Panama Street  
Address: Pine Hall, room 125  
City: Stanford  
StateProv: CA  
PostalCode: 94305-4102  
Country: US  
RegDate: 2010-01-05  
Updated: 2020-03-27  
Ref: https://rdap.arin.net/registry/entity/STANF0-Z

Tracert stanford.edu:



The screenshot shows a terminal window titled 'shahdyahya — zsh — 106x24'. The terminal output shows the results of a traceroute to stanford.edu (171.67.215.200). The output is as follows:

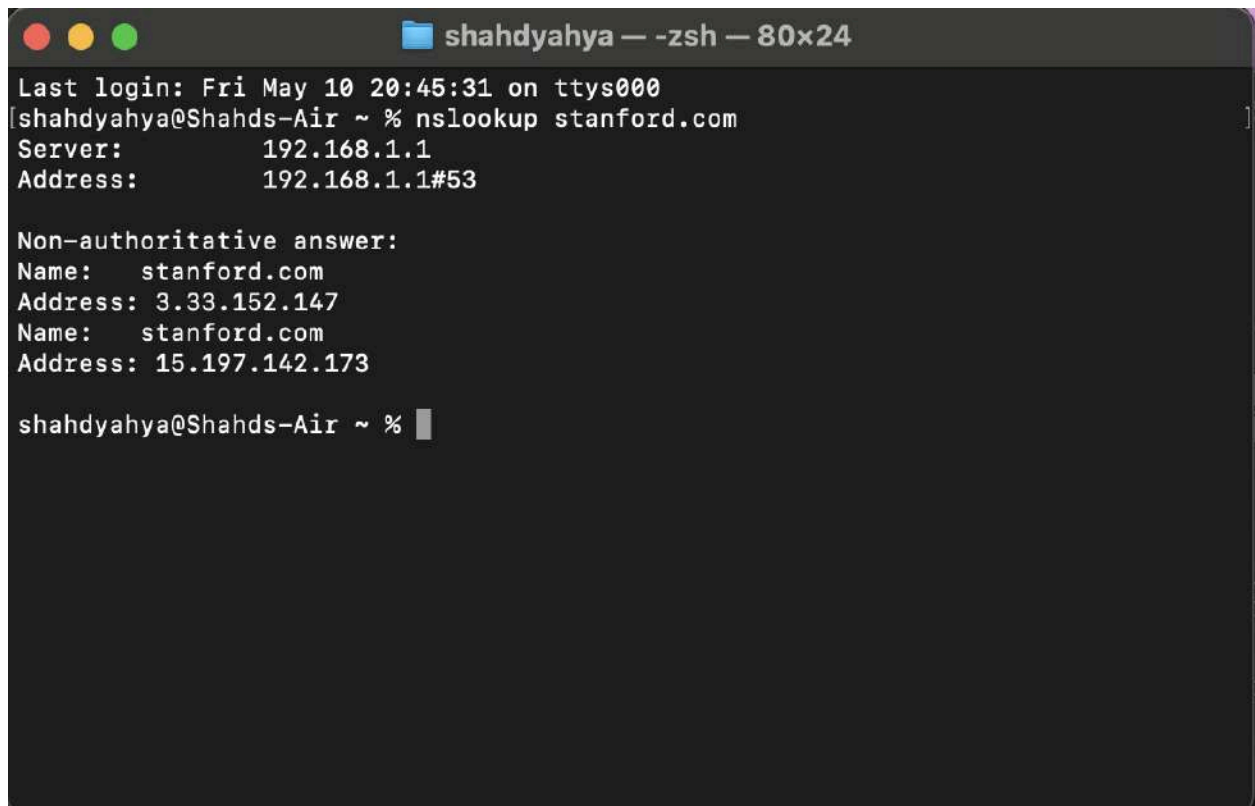
```
Last login: Fri May 10 20:44:44 on ttys000
shahdyahya@Shahds-Air ~ % traceroute stanford.edu
traceroute to stanford.edu (171.67.215.200), 64 hops max, 40 byte packets
 1  superfast (192.168.1.1)  11.467 ms  4.611 ms  5.646 ms
 2  10.74.32.246 (10.74.32.246)  24.412 ms  21.795 ms  18.777 ms
 3  10.74.19.113 (10.74.19.113)  62.012 ms  63.676 ms  *
 4  10.74.19.150 (10.74.19.150)  62.545 ms  61.098 ms  61.308 ms
 5  10.74.59.186 (10.74.59.186)  60.443 ms  72.128 ms  68.340 ms
 6  et-4-0-13.edge1.marseille3.level3.net (213.242.111.121)  86.434 ms  75.289 ms  60.478 ms
 7  ae1.3505.edge9.sanjose1.level3.net (4.69.219.61)  323.450 ms  *  *
 8  cenic.edge9.sanjose1.level3.net (4.15.122.46)  223.826 ms  224.404 ms  222.017 ms
 9  dc-snv12-agg-01--sv1-agg10-400g.cenic.net (137.164.11.81)  217.804 ms  221.894 ms  *
10  dc-stanford--snv12-agg-01-100ge.cenic.net (137.164.23.145)  353.787 ms  305.968 ms  307.215 ms
11  campus-ial-nets-b-v11100.sunet (171.66.255.192)  233.565 ms
    campus-nw-rtr-v11000.sunet (171.64.255.192)  233.443 ms  239.043 ms
12  * * *
13  web.stanford.edu (171.67.215.200)  363.908 ms  224.964 ms  286.065 ms
shahdyahya@Shahds-Air ~ %
```

It traces the routes your data packets take to reach Stanford, listing each router (identified by IP or hostname) along the way, and it shows the time it took for a packet to reach that router and return a response, measured in milliseconds (ms).

The initial hops (1-5) are likely within your local network, showing relatively low round-trip times (around 10-70 ms), then the times increase significantly (over 200 ms), indicating the packets are traveling long distances over the internet.

The Asterisks (\*) means that there was no response received from that router.

The final line (13) shows the destination (web.stanford.edu) with its IP address (171.67.215.200) and the corresponding round-trip times

A terminal window titled "shahdyahya — -zsh — 80x24" with standard macOS window controls. The terminal shows the output of the command "nslookup stanford.com". It displays the server IP (192.168.1.1) and the address (192.168.1.1#53). Below this, it shows a "Non-authoritative answer:" with two entries for "stanford.com": one with IP 3.33.152.147 and another with IP 15.197.142.173. The prompt "shahdyahya@Shahds-Air ~ %" is visible at the bottom.

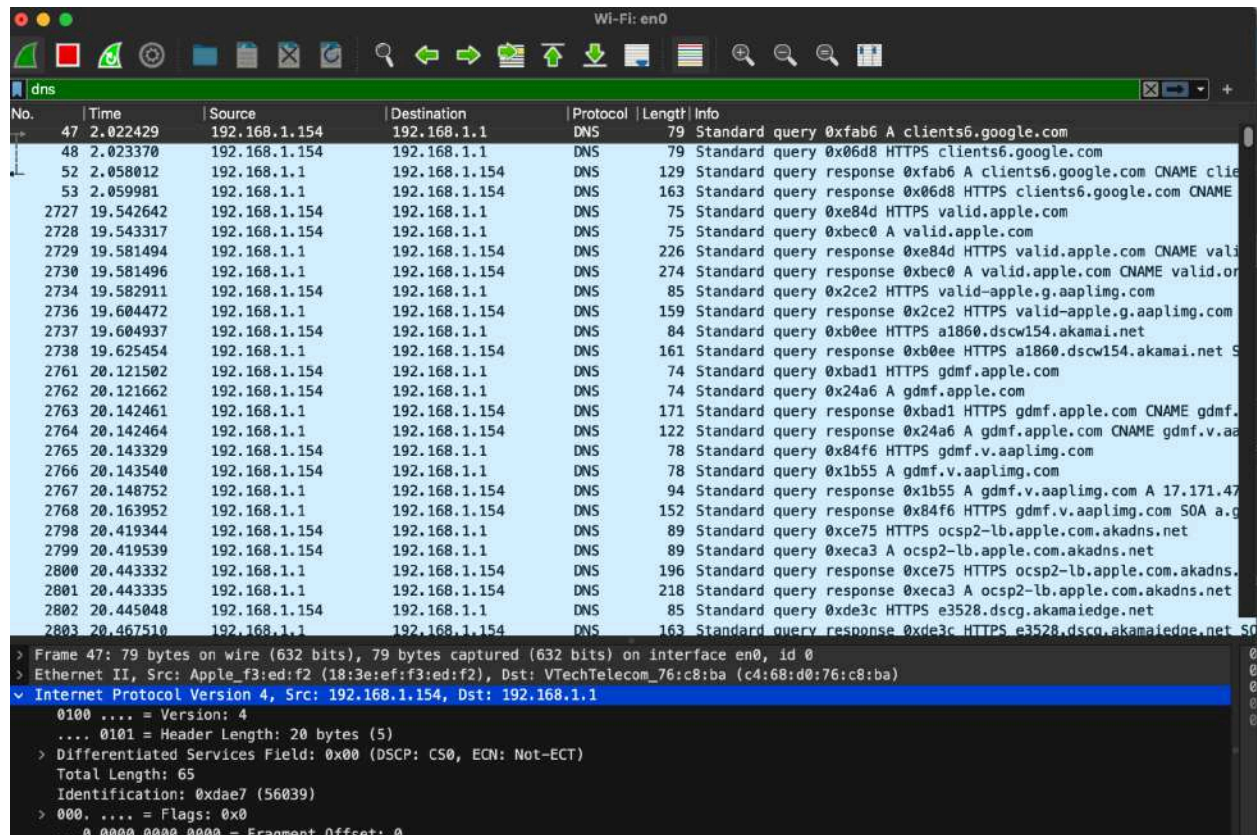
```
shahdyahya@Shahds-Air ~ % nslookup stanford.com
Server:          192.168.1.1
Address:         192.168.1.1#53

Non-authoritative answer:
Name:   stanford.com
Address: 3.33.152.147
Name:   stanford.com
Address: 15.197.142.173

shahdyahya@Shahds-Air ~ %
```



I have used wireshark to capture dns results:



The screenshot displays a Wireshark capture of DNS traffic. The main pane shows a list of packets with columns for No., Time, Source, Destination, Protocol, Length, and Info. The packets are filtered by 'dns'. The packet list shows a series of DNS queries and responses. The packet details pane for packet 47 is expanded, showing the Internet Protocol Version 4 header and the DNS query details.

No.	Time	Source	Destination	Protocol	Length	Info
47	2.022429	192.168.1.154	192.168.1.1	DNS	79	Standard query 0xfab6 A clients6.google.com
48	2.023370	192.168.1.154	192.168.1.1	DNS	79	Standard query 0x06d8 HTTPS clients6.google.com
52	2.058012	192.168.1.1	192.168.1.154	DNS	129	Standard query response 0xfab6 A clients6.google.com CNAME clie
53	2.059981	192.168.1.1	192.168.1.154	DNS	163	Standard query response 0x06d8 HTTPS clients6.google.com CNAME
2727	19.542642	192.168.1.154	192.168.1.1	DNS	75	Standard query 0xe84d HTTPS valid.apple.com
2728	19.543317	192.168.1.154	192.168.1.1	DNS	75	Standard query 0xbec0 A valid.apple.com
2729	19.581494	192.168.1.1	192.168.1.154	DNS	226	Standard query response 0xe84d HTTPS valid.apple.com CNAME vali
2730	19.581496	192.168.1.1	192.168.1.154	DNS	274	Standard query response 0xbec0 A valid.apple.com CNAME valid.or
2734	19.582911	192.168.1.154	192.168.1.1	DNS	85	Standard query 0x2ce2 HTTPS valid-apple.g.aapling.com
2736	19.604472	192.168.1.1	192.168.1.154	DNS	159	Standard query response 0x2ce2 HTTPS valid-apple.g.aapling.com
2737	19.604937	192.168.1.154	192.168.1.1	DNS	84	Standard query 0xb0ee HTTPS a1860.dscw154.akamai.net
2738	19.625454	192.168.1.1	192.168.1.154	DNS	161	Standard query response 0xb0ee HTTPS a1860.dscw154.akamai.net S
2761	20.121502	192.168.1.154	192.168.1.1	DNS	74	Standard query 0xbad1 HTTPS gdmf.apple.com
2762	20.121662	192.168.1.154	192.168.1.1	DNS	74	Standard query 0x24a6 A gdmf.apple.com
2763	20.142461	192.168.1.1	192.168.1.154	DNS	171	Standard query response 0xbad1 HTTPS gdmf.apple.com CNAME gdmf.
2764	20.142464	192.168.1.1	192.168.1.154	DNS	122	Standard query response 0x24a6 A gdmf.apple.com CNAME gdmf.v.a
2765	20.143329	192.168.1.154	192.168.1.1	DNS	78	Standard query 0x84f6 HTTPS gdmf.v.aapling.com
2766	20.143540	192.168.1.154	192.168.1.1	DNS	78	Standard query 0x1b55 A gdmf.v.aapling.com
2767	20.148752	192.168.1.1	192.168.1.154	DNS	94	Standard query response 0x1b55 A gdmf.v.aapling.com A 17.171.47
2768	20.163952	192.168.1.1	192.168.1.154	DNS	152	Standard query response 0x84f6 HTTPS gdmf.v.aapling.com SOA a.g
2798	20.419344	192.168.1.154	192.168.1.1	DNS	89	Standard query 0xce75 HTTPS ocsp2-lb.apple.com.akadns.net
2799	20.419539	192.168.1.154	192.168.1.1	DNS	89	Standard query 0xeca3 A ocsp2-lb.apple.com.akadns.net
2800	20.443332	192.168.1.1	192.168.1.154	DNS	196	Standard query response 0xce75 HTTPS ocsp2-lb.apple.com.akadns.
2801	20.443335	192.168.1.1	192.168.1.154	DNS	218	Standard query response 0xeca3 A ocsp2-lb.apple.com.akadns.net
2802	20.445048	192.168.1.154	192.168.1.1	DNS	85	Standard query 0xde3c HTTPS e3528.dscg.akamaiedge.net
2803	20.467510	192.168.1.1	192.168.1.154	DNS	163	Standard query response 0xde3c HTTPS e3528.dscg.akamaiedge.net SO

Frame 47: 79 bytes on wire (632 bits), 79 bytes captured (632 bits) on interface en0, id 0  
> Ethernet II, Src: Apple\_f3:ed:f2 (18:3e:ef:f3:ed:f2), Dst: VTEchTelecom\_76:c8:ba (c4:68:d0:76:c8:ba)  
Internet Protocol Version 4, Src: 192.168.1.154, Dst: 192.168.1.1  
0100 .... = Version: 4  
.... 0101 = Header Length: 20 bytes (5)  
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)  
Total Length: 65  
Identification: 0xdae7 (56039)  
> 000. .... = Flags: 0x0  
... 0 0000 0000 0000 = Fragment Offset: 0

The screenshot shows a series of DNS queries made from a computer with the IP address 192.168.1.154 to a DNS server with the IP address 192.168.1.1.

## Part 2:

In this part I implemented a server and a client ,the server deliver the messages from a clients and connect the peer to each other ( by sending the new ports it received to the all peers so that they can send and deliver from each other)

Here is the code of the server:

```
1 import socket
2 import threading
3 import time
4
5 host = '127.0.0.1'
6 port = 5051
7 peers = ['5051'] # add the peers that sent messages to the client
8 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #create a socket
9 client = {}
10 temp = {}
11 i = 1
12 usage
13
14 def receive_messages():
15     global i # Access the global variable i
16     while True:
17         data, addr = sock.recvfrom(1024)
18         message = data.decode()
19         # if the peer send "CONNECT" to the server then we can add it to the peer array
20         if message == "CONNECT":
21             peers.append(str(addr[1]))
22             broadcast()
23         # if the peer send the message to the server we display it
24         else:
25             current_time = time.strftime('%H:%M:%S', time.localtime())
26             words = message.split()
27             Message = " ".join(words[2:]) # Join remaining words into Message
28             # Store client details in a tuple and add to client dictionary
29             client[addr[1]] = (words[0], words[1], current_time, Message)
30             # Store address in temp list using current index i
31             temp[i] = addr[1]
32             # Print received message information
33             print(f'{i} - Received message from {words[0]} {words[1]} at {current_time}')
34             # Increment i to prepare for the next message
35             i += 1
36     --
37
38 usage
39
40 def broadcast():
41     peersMsg = ",".join(peers) # Join peers into a comma-separated string
42     for peer in peers:
43         if int(peer) != port: # do not send the port to the server
44             sock.sendto(peersMsg.encode(), (host, int(peer))) # send the new peer port to all the peers so that they can send it to the peers
45
46
47 if __name__ == "__main__":
48     sock.bind((host, port))
49     print(f"Server listening on {host}:{port}")
50     # Start a thread to continuously receive messages
51     threading.Thread(target=receive_messages).start()
52     print("Peer first name last name")
53
54     while True:
55         if i == 3:
56             option = input("Enter an option: ")
57             if int(option) in range(1, i + 1):
58                 first_name, second_name, received_time, message = client[temp[int(option)]]
59                 print("Your message is:", message)
60             else:
61                 print("Invalid option")
```

In this code I bind the socket to the host and port (5051). And Starts a thread to run receive\_messages continuously.then it asks the user to enter first ,last name and the message.

In the threaded function it continuously listens for incoming messages using recvfrom and If the client sends "CONNECT" it adds the client's port to the peers list and broadcasts Otherwise, extracts message details, stores them in the client.

In the broadcast function it iterates through peers and sends the updated list (except the server's port) to connected clients.This allows clients to discover other connected peers.

Here is the code of the client:

```
1 import socket
2 import threading
3 import time
4
5 host = '127.0.0.1'
6 peers = []
7 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
8 client = {}
9 temp = {}
10 server_port = 5051
11 i = 1
12
13 usage
14 def receive_messages():
15     global i # Access the global variable i
16     while True:
17         data, addr = sock.recvfrom(1024)
18         message = data.decode()
19         # If the port is the server port then add the message holds the new ports of the new peers
20         # so we add it the targeted peers
21         if addr[1] == server_port:
22             processServerMsg(message)
23
24         else: # If it is not from the server then it is from one of the peers so we extract a message from it
25             current_time = time.strftime(format: '%H:%M:%S', time.localtime())
26             words = message.split()
27             Message = " ".join(words[2:]) # Join remaining words into Message
28             # Store client details in a tuple and add to client dictionary
29             client[addr[1]] = (words[0], words[1], current_time, Message)
30             # Store address in temp list using current index i
31             temp[i] = addr[1]
32             # Increment i to prepare for the next message
33             i += 1
```



The rest of the code :

```
34 def broadcast(message): # send the message to all the peers
35     global peers
36
37     for peer in peers:
38         if peer != port: # Don't send to myself
39             sock.sendto(message.encode(), (host, peer))
40
41 usage
42 def talkToServer():
43     sock.sendto("CONNECT".encode(), (host, server_port))
44
45 usage
46 def processServerMsg(message): #add the new peers sent by the servers to the peerslist
47     global peers
48     newPeers = message.split(",")
49     peers = []
50     for newPeer in newPeers:
51         if newPeer == "":
52             continue
53         peers.append(int(newPeer))
54
55 if __name__ == "__main__":
56     # Bind the socket to an available port on the host
57     sock.bind((host, 0))
58     port = sock.getsockname()[1]
59     print(f"Peer listening on {host}:{port}")
60     # Start a thread to continuously receive messages
61     talkToServer()
62     threading.Thread(target=receive_messages).start()
63
64     # Get user input for first name, last name, and message
65     firstName = input("Enter your first name: ")
66     lastName = input("Enter your last name: ")
67     message = input("Enter message: ")
68     # Construct the message to be broadcasted
69     Message = f'{firstName} {lastName} {message}'
70     # Broadcast the message to all peers
71     broadcast(Message)
72
73     while i != 2:
74         time.sleep(1) # Wait for 1 second before checking again
75         print('Peer first name last name')
76
77     for j in temp:
78         first_name, second_name, received_time, message = client[temp[j]]
79         print(f'{j} - Received message from {first_name} {second_name} at {received_time}')
80
81     while True:
82         option = input("Enter an option: ")
83         if int(option) in range(1, i + 1):
84             first_name, second_name, received_time, message = client[temp[int(option)]]
85             print("Your message is:", message)
86         else:
87             print("Invalid option")
```

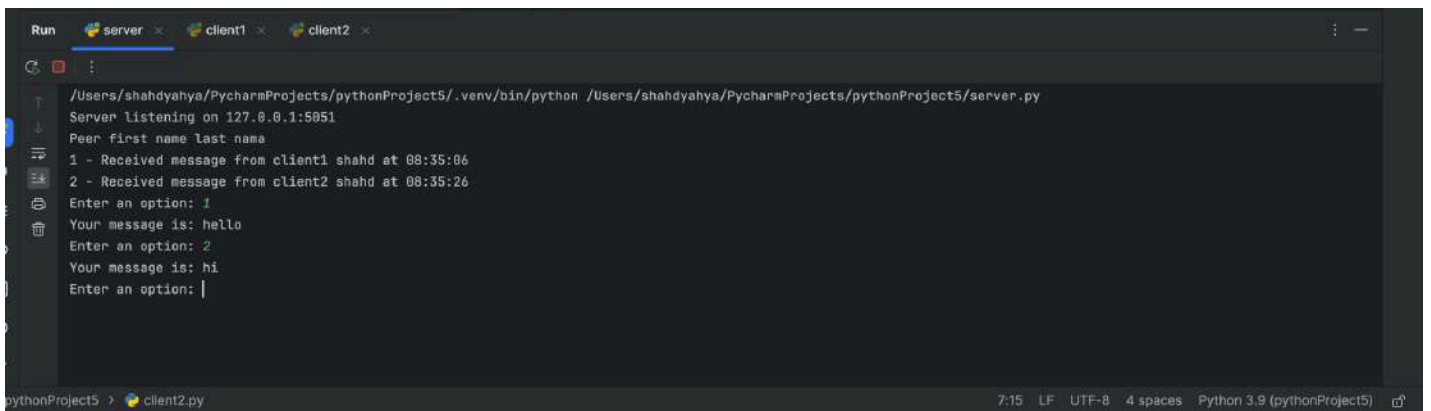
In this code I bind the socket to an available port on the local machine.and then starts a thread for receive\_messages to listen for messages continuously.then Prompts the user to enter their name and a message.and Broadcasts the message to all connected peers using the broadcast function.

In the threaded function `receive_messages` it listens for incoming messages. If the message comes from the server port (5051), it means that the server sent a new peer port and calls `processServerMsg` to update the list of connected peers. Otherwise, it extracts message details.

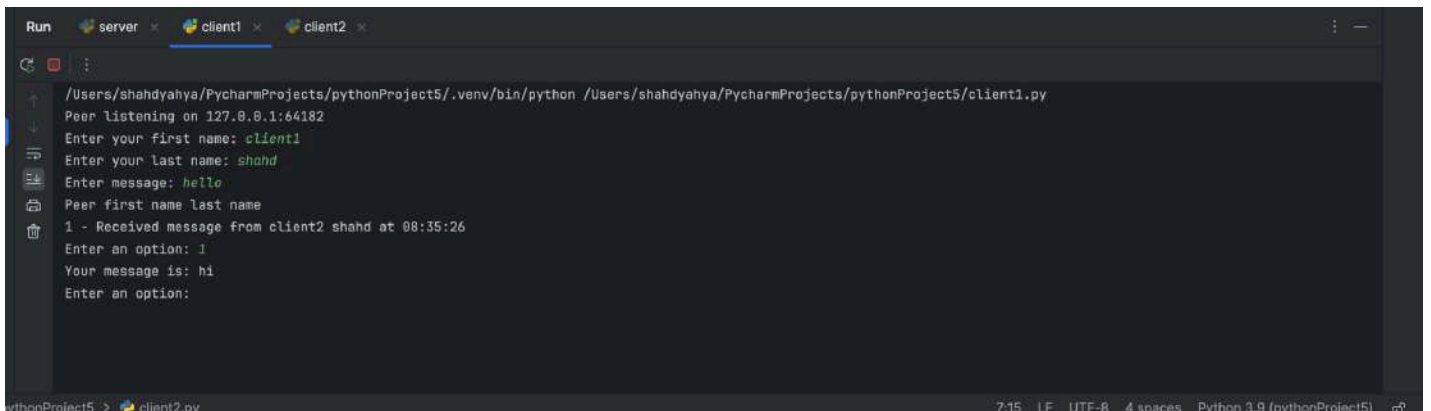
In the broadcast function it iterates through the peers list and sends the message to each connected client.

In the `talkToServer` function it sends a "CONNECT" message to the server to announce the client's presence.

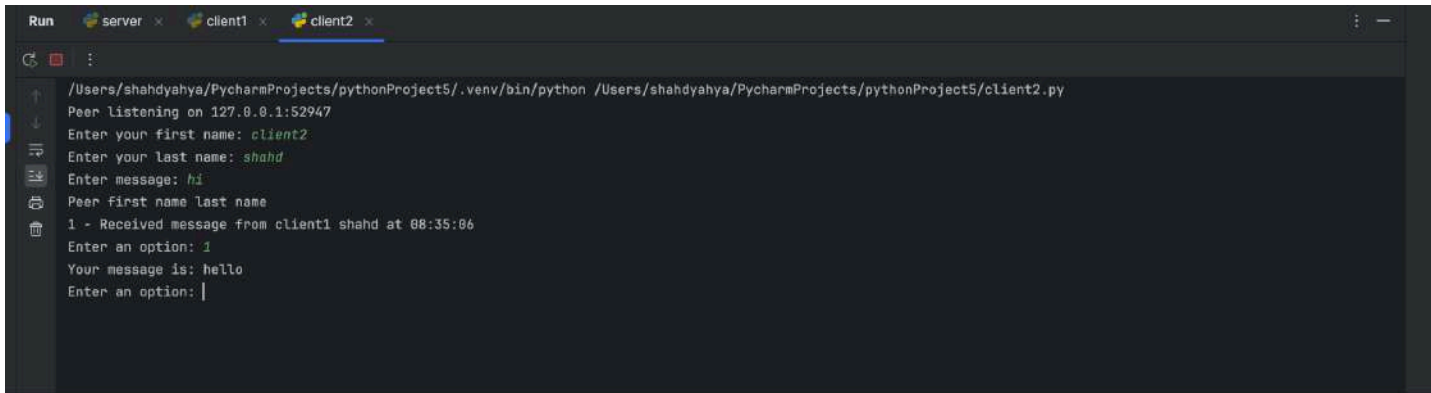
Here is the output of a server and 2 clients:



```
Run server client1 client2
/Users/shahdyahya/PycharmProjects/pythonProject5/.venv/bin/python /Users/shahdyahya/PycharmProjects/pythonProject5/server.py
Server listening on 127.0.0.1:5051
Peer first name last name
1 - Received message from client1 shahd at 08:35:06
2 - Received message from client2 shahd at 08:35:26
Enter an option: 1
Your message is: hello
Enter an option: 2
Your message is: hi
Enter an option: |
```



```
Run server client1 client2
/Users/shahdyahya/PycharmProjects/pythonProject5/.venv/bin/python /Users/shahdyahya/PycharmProjects/pythonProject5/client1.py
Peer listening on 127.0.0.1:64182
Enter your first name: client1
Enter your last name: shahd
Enter message: hello
Peer first name last name
1 - Received message from client2 shahd at 08:35:26
Enter an option: 1
Your message is: hi
Enter an option:
```



```
Run server client1 client2
/Users/shahdyahya/PycharmProjects/pythonProject5/.venv/bin/python /Users/shahdyahya/PycharmProjects/pythonProject5/client2.py
Peer listening on 127.0.0.1:52947
Enter your first name: client2
Enter your last name: shahd
Enter message: hi
Peer first name last name
1 - Received message from client1 shahd at 08:35:06
Enter an option: 1
Your message is: hello
Enter an option: |
```

## Part 3:

**Entity tag cache validators** : are unique identifiers that act as a more dependable way to validate cached content. This improves efficiency by letting browsers reuse cached content when it truly hasn't changed, reducing traffic and server load.

Here is the code of my program:

```
1  from socket import *
2
3  serverPort = 6060
4  serverSocket = socket(AF_INET, SOCK_STREAM)
5  serverSocket.bind(('', serverPort))
6  serverSocket.listen(1)
7  print("The server is ready to receive")
8  while True:
9      connectionSocket, addr = serverSocket.accept()
10     ip = addr[0]
11     port = addr[1]
12     sentence = connectionSocket.recv(1024).decode()
13     request = sentence.split()[1]
14     print("the request is : " + request)
15     if (request == '/' or request == '/index.html' or request == '/main_en.html' or request == '/en'): # The if statement checks whether the requested object is
16         connectionSocket.send("HTTP/1.1 200 OK \r\n".encode())
17         connectionSocket.send("Content-Type: text/html \r\n".encode())
18         connectionSocket.send("\r\n".encode())
19         fileminen = open("main_en.html", "rb")
20         connectionSocket.send(fileminen.read()) # read the file that was open when it called
21
22     elif (request == '/ar'): # The if statement checks whether the requested object is one of several specific values
23         connectionSocket.send("HTTP/1.1 200 OK \r\n".encode())
24         connectionSocket.send("Content-Type: text/html \r\n".encode())
25         connectionSocket.send("\r\n".encode())
26         fileminen = open("main_ar.html", "rb")
27         connectionSocket.send(fileminen.read()) # read the file that was open when it called
28
29     elif (request.endswith('.html')):
30         connectionSocket.send("HTTP/1.1 200 OK \r\n".encode())
31         connectionSocket.send("Content-Type: text/html \r\n".encode())
32         connectionSocket.send("\r\n".encode())
33         filelink = open("myform.html", "rb")
34         connectionSocket.send(filelink.read())
35
36     elif (request.endswith('.css')):
37         connectionSocket.send("HTTP/1.1 200 OK \r\n".encode())
38         connectionSocket.send("Content-Type: text/css \r\n".encode())
39         connectionSocket.send("\r\n".encode())
40         filecss = open("style.css", "rb")
41         connectionSocket.send(filecss.read())
42
43     elif (request.endswith('.png')): # files with the extensions '.png' and '.jpg'.
44         connectionSocket.send("HTTP/1.1 200 OK \r\n".encode())
45         connectionSocket.send("Content-Type: image/png \r\n".encode())
46         connectionSocket.send("\r\n".encode())
47         filepngimg = open("image3.png", "rb")
48         connectionSocket.send(filepngimg.read())
49
50     elif (request.endswith('.jpg')): # The same process occurs for '.jpg' files.
51         connectionSocket.send("HTTP/1.1 200 OK \r\n".encode())
52         connectionSocket.send("Content-Type: image/jpg \r\n".encode())
53         connectionSocket.send("\r\n".encode())
54         filejpgimg = open("image4.jpg", "rb") # open the image with jpg extension.
55         connectionSocket.send(filejpgimg.read())
56
57     elif (request == '/so'):
58
59         connectionSocket.send("HTTP/1.1 307 Temporary Redirect \r\n".encode())
60         connectionSocket.send("Content-Type: text/html \r\n".encode())
61         connectionSocket.send("Location: https://stackoverflow.com \r\n".encode())
62         connectionSocket.send("\r\n".encode())
```

The rest of the code:

```
64 elif (request == '/itc'):
65     connectionSocket.send("HTTP/1.1 307 Temporary Redirect \r\n".encode())
66     connectionSocket.send("Content-Type: text/html \r\n".encode())
67     connectionSocket.send("Location: https://itc.birzeit.edu \r\n".encode())
68     connectionSocket.send("\r\n".encode())
69
70
71 else: # scenario where a requested resource is not found by a client.
72     connectionSocket.send("HTTP/1.1 404 Not Found \r\n".encode())
73     connectionSocket.send("Content-Type: text/html \r\n".encode())
74     connectionSocket.send("\r\n".encode())
75     notFoundHtmlString = "<html>" \
76         "<head>" \
77         "<title>ERROR 404 </title>" \
78         "</head>" \
79         "<body>" \
80         "<pre>" \
81         "<p style='font-size: 45px; text-align:center; color:Red;'>" \
82         "The file is not found </p>" \
83         "<p style='font-size: 25px; text-align:center; color:Black;'>" \
84         "<b> Shahd Yahya 1210249 <b/></p>" \
85         "</pre>" \
86         "<pre style='font-size: 25px; text-align:center;'>" \
87         f"The IP is: {ip}" \
88         f"The port is: {port}" \
89         "</pre>" \
90         "</body>" \
91         "</html>"
92     notFoundHtmlBytes = bytes(notFoundHtmlString, "UTF-8")
93     connectionSocket.send(notFoundHtmlBytes)
94
95
```

First I defined the server port (6060).then created a TCP socket (serverSocket) for listening on this port.then binds the socket to the port and starts listening for incoming connections.

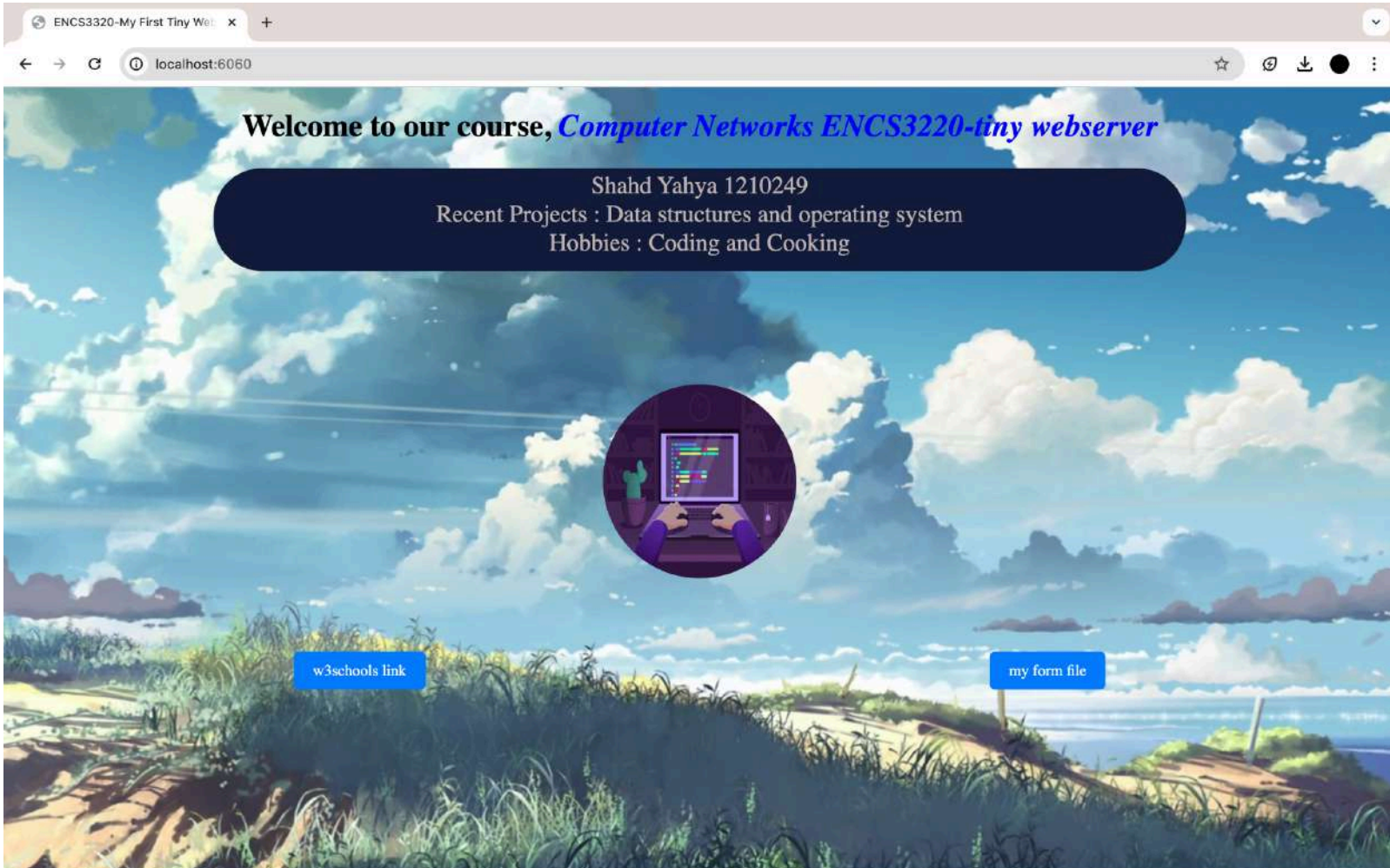
When a client connects, serverSocket.accept() accepts the connection and returns a new socket and it receives the request message sent by the client and decodes it.

- If the requested object is main\_en.html, index.html, or similar, it sends a 200 OK response with the content of "main\_en.html".
- If the requested object ends with ".html" (but isn't a specific file mentioned before), it sends a 200 OK response with the content of "myform.html".
- If it ends with ".css", it sends a 200 OK response with the content of "style.css" (assuming a CSS file).
- If it ends with ".png", it sends a 200 OK response with the content of "image3.png" (assuming a PNG image).
- If it ends with ".jpg", it sends a 200 OK response with the content of "image4.jpg" (assuming a JPG image).
- If the requested object is "/so", it sends a 307 Temporary Redirect response to redirect the client to Stack Overflow.

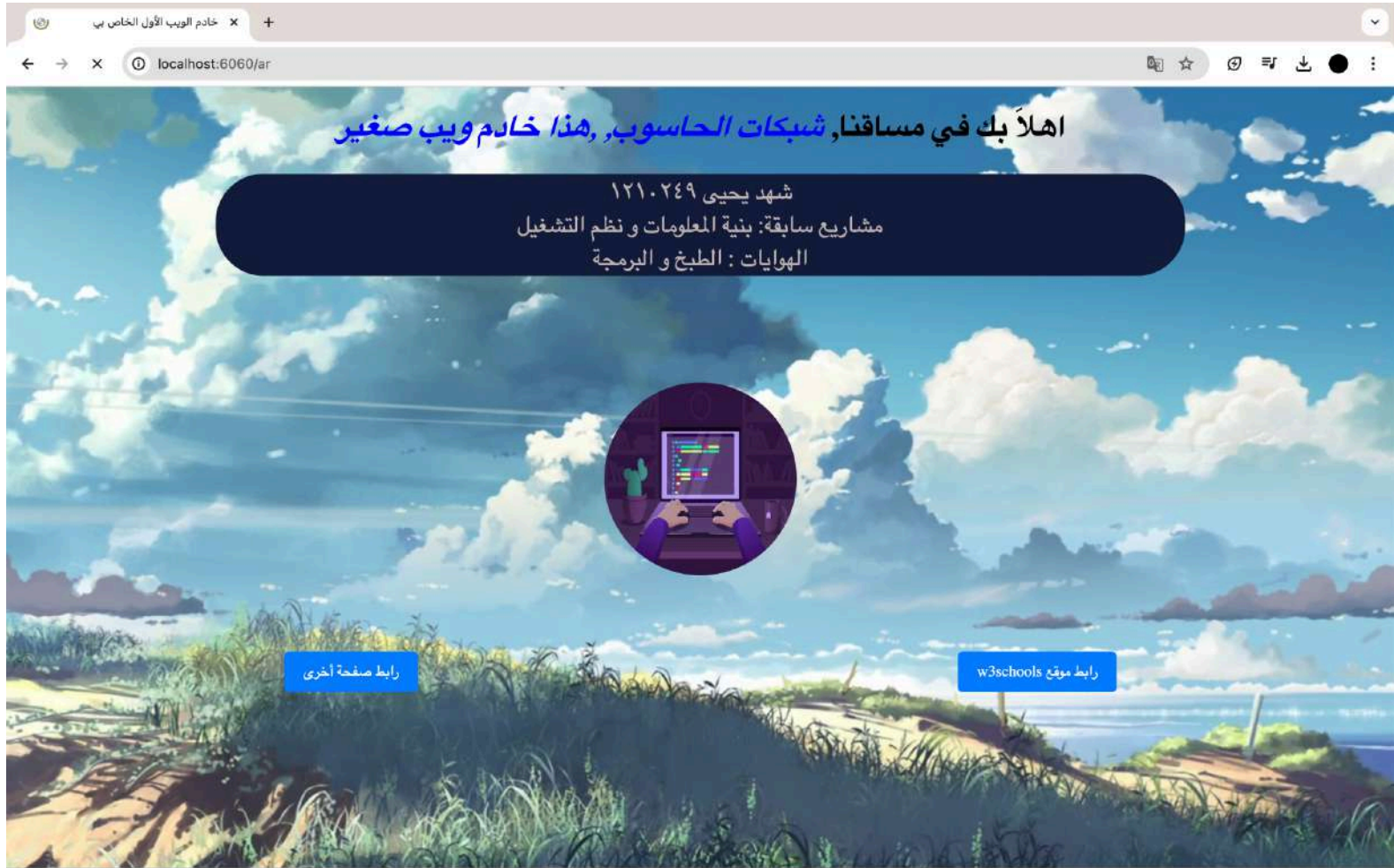


- If the requested object is `"/itc"`, it sends a 307 Temporary Redirect response to redirect the client to Birzeit University.
- If none of the above match (file not found), it sends a 404 Not Found response with a custom error message that includes the client's IP and port.

The web server :







```

/Users/shahdyahya/PycharmProjects/pythonProject2/.venv/bin/python /Users/shahdyahya/PycharmProjects/pythonProject2/part3.py
The server is ready to receive
GET / HTTP/1.1
Host: localhost:6060
Connection: keep-alive
sec-ch-ua: "Chromium";v="124", "Google Chrome";v="124", "Not-A.Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36
Sec-Purpose: prefetch;prerender
Purpose: prefetch
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
Cookie: Pycharm-4e3315b=ae8d0834-2e92-43a6-9d3b-28349f7c28d0

the request is : /
GET /style.css HTTP/1.1
Host: localhost:6060
Connection: keep-alive
sec-ch-ua: "Chromium";v="124", "Google Chrome";v="124", "Not-A.Brand";v="99"
Sec-Purpose: prefetch;prerender
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36
sec-ch-ua-platform: "macOS"
Accept: text/css,*/*;q=0.1

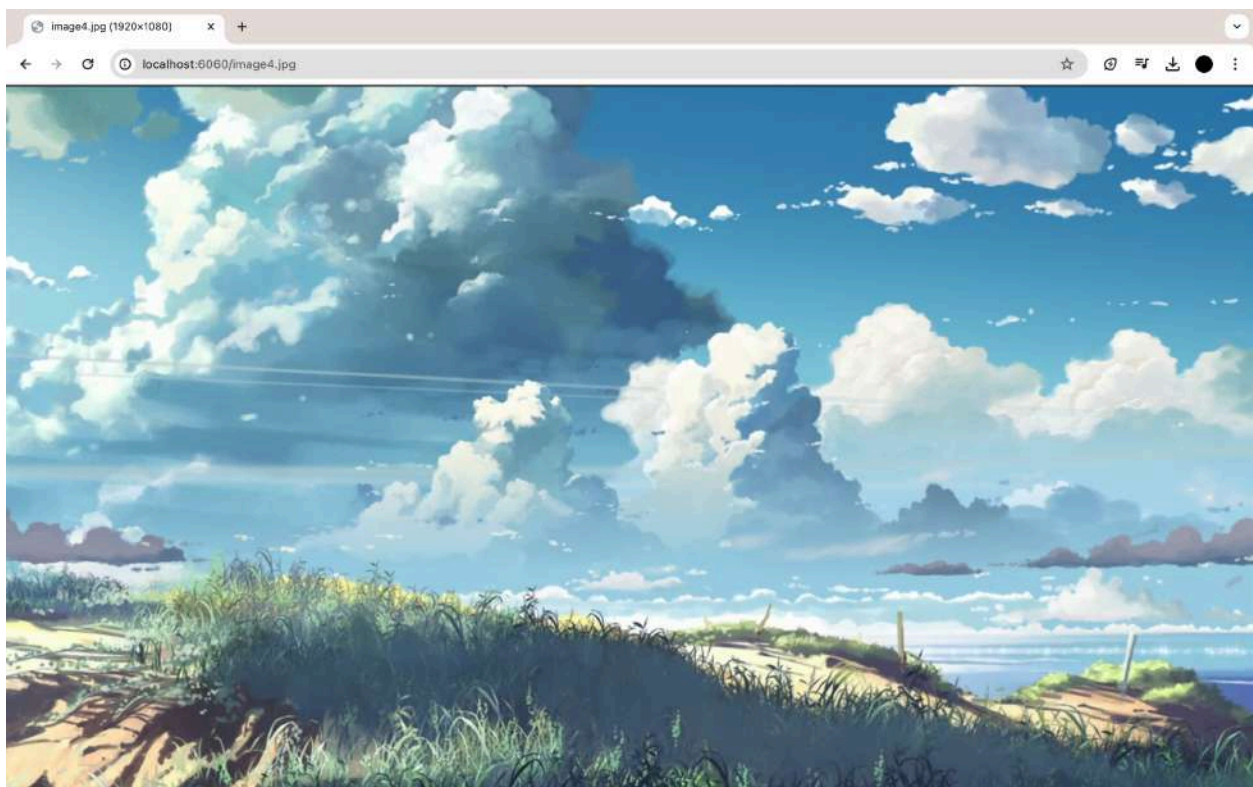
```

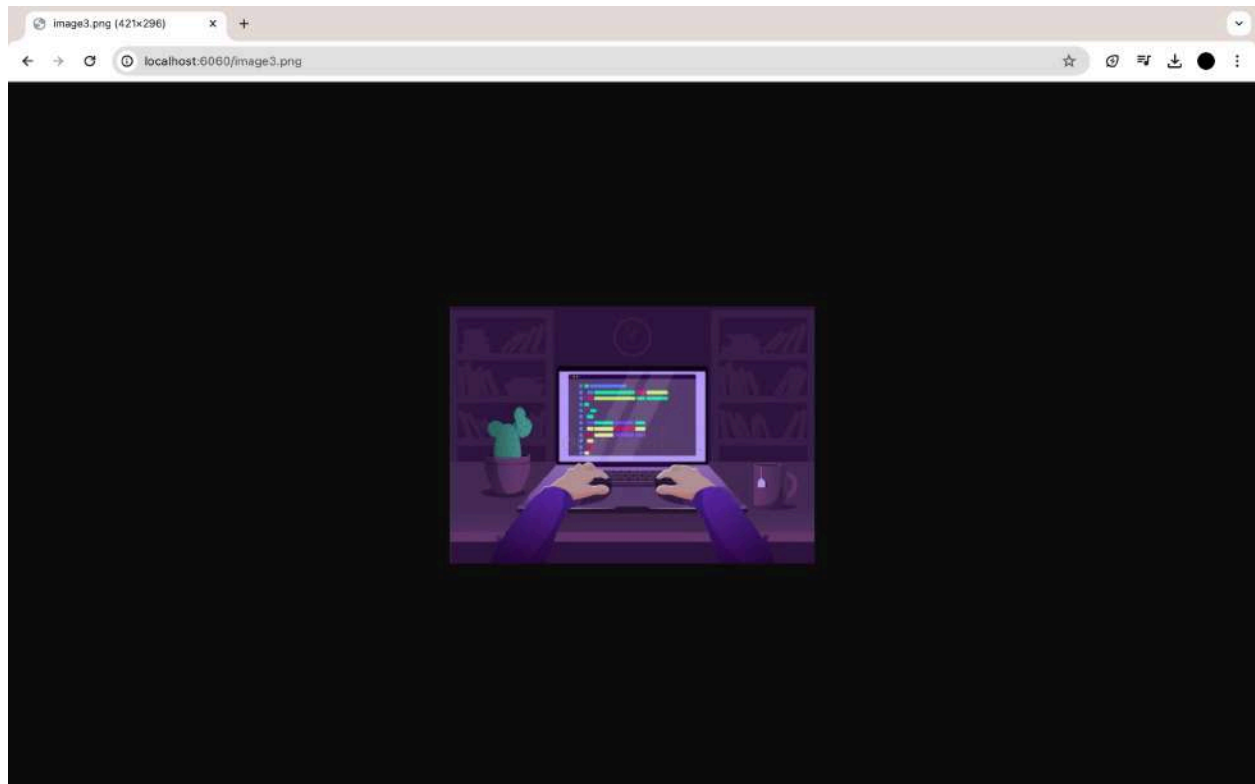


The file is not found

Shahd Yahya 1210249

The IP is: 127.0.0.1      The port is: 54670





```
body { /* Background pic */
  background-image: url("image4.jpg");
  background-position: center top;
  background-repeat: no-repeat;
  background-size: cover;
}

h1 {
  text-align: center;
}

em {
  color: blue;
}

.container1 {
  background-color: #131d3b; /* boxes of names and hobbies */
  border-radius: 50px;
  color: #c2b3b3;
  font-family: serif;
  font-size: 130%;
  padding: 30px;
  margin: 25px auto;
  width: 1000px;
  height: 100px;
  text-align: center;
}

p {
  font-size: 25px;
  margin: 0 auto;
}

img {
  width: 200px;
  height: 200px;
  border-radius: 100px;
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%); /* Center the image using transform */
}

.container2 {
  display: flex;
  justify-content: center;
  align-items: flex-end;
  min-height: 50vh;
  padding-bottom: 20px;
}

a.a, a.b {
```



From other device:

